REMARKS

This Amendment is responsive to the Office Action dated July 13, 2007, in this application. Claims 1, 4-10 and 12-24 are amended hereby. Claims 1-24 remain pending hereinafter, where claims 1, 10 and 18 are the independent claims.

Provisional Rejection of claims 6, 14 and 21 Under Non-Statutory Obviousness-Type Double Patenting Doctrine

In the July 13, 2007 Office Action, the Examiner rejects claims 6, 14 and 21 under the judicially created doctrine of obviousness-type double patenting over claims 1, 18 of co-pending, commonly owned application Serial No. 11/227,761 (the '761 application").

With respect to claim 6, the Examiner states that while claim 1 of the '761 application is not identical, claim 6 recites dynamically updating an operating system by identifying references to said first code component and replacing the identified references to said first code component to said new code component. The Examiner further states that cited claim 1 of the '761 application does not explicitly recite separating the first code component into objects grouped in table, but that one skilled in the art would infer that the reference pointer could be changed to point to a new object instead of a factory configured object, and removing the factory object, and therefore provide said runtime (new) factory object in form of runtime table entry as in table storing reference entries so that the table-represented reference pointers identify during the dynamic replacement of reference objects in view of well-known approach using reference table to interrelate dynamic code referencing to provide program resolution support at runtime.

With respect to claims 6, 14 and 21, the Examiner indicates that claim 18 of the '761 application recites updating an operating system without rebooting, identifying references to said first code component and replacing the identified references to said first code component to said new code component, but that the '761 application does not explicitly recite by what is described by the Examiner as separating the first code component into objects grouped in table, whereby references to said objects are entered in the table, but that in view of the structure to store the definition referring to the object instance being swapped from above, this reference table limitation would have been obvious.

Applicants respectfully disagree that claims 6, 14 and 21 are patentably indistinct from claims 1 and 18 of the '761 application.

A non-statutory obviousness-type double patenting rejection is appropriate where the conflicting claims are either anticipated by, or would have been obvious over the referenced claims. MPEP 804; In re Berg, 140 F.3d 1428, 46 USPQ2d 1226 (Fed. Cir. 1998). Such rejections are in order where the claimed subject matter is not patentable distinct from the cited claimed subject matter.

Applicants respectfully assert that the subject matter of claims 6, 14 and 21 is patentably distinct from the subject matter of independent claims 1 and 18 of the '761 application. The inventions set forth in claims 6, 14 and 21 are the inventions of independent claims 3, 13 and 1, respectively, modified by the limitations set forth in the dependent claims. Because the subject matter of independent claims 3, 13 and 1 is patentably distinguishable from the subject matter of claims 1 and 18 of the '761 invention, the subject matter of the dependent claims that derive from those independent claims, i.e., dependent claims 6, 14 and 21, is patentably distinguishable from claims 1 and 18 of the '761 application for at least the reasons that distinguish independent claims 3, 13 and 1 from same claims 1, and 18 of the '761 application. Applicants therefore request withdrawal of the rejection of claims 6, 14 and 21 under the doctrine of obviousness-type double patenting.

Objections To The Specification and Claims

The Specification is objected to at paragraph 4 of the Office Action. Claims 1-24 are objected to for an informality that paragraph [0013] is said to describe hot swapping in an operating system to replace first source code components with new source code components, the Examiner asserting that there is insufficient teaching throughout the disclosure to support (insufficient enablement as to) how source code components are replaced in the hot swap using multi-processor environment and state safe redirecting live threads, absent any further mention of source code elsewhere. The Examiner indicates that correction to this source code in the disclosed method of hot swapping is required.

The Examiner further objects to claims 1-24 at paragraph 5 of the Office action, because of the terms "first source code" and "new source code" in that OS runtime context are non-enabling, and that they need to be corrected because no source code is part of the runtime replacement and indirection process to secure a thread save continual execution. The Examiner suggests that source code should be treated as mere code instances being source to a replacement step whereby more instances of code are derived.

In response, applicants have amended paragraph [0013] and claims 1-24 to refer to the components as first code component and new code component, no longer using the qualifier "source," and respectfully request withdrawal of the objections to the Specification, and claims 1-24, as asserted at paragraphs 4 and 5 of the Office Action.

Claims 5, 7-9, 13, 15-17, 20 and 22-24 were objected to at paragraph 6 of the Office action. The Examiner asserts that there is no description in the Specification that explicitly conveys transferring of a quiescent state per se, which is disclosed as being established or achieved, and that subsequent thereto a transfer is done and relates to a "state of a call" or an object – no quiescent state being transferred.

In response, applicants have amended the objected to claims to identify that the first code component references at the quiescent state are identified, the same identified (quiescent state) references are transferred to the new code component, and thereafter, the first code component is swapped for the new code component. Applicants, therefore, respectfully request withdrawal of the objections to claims 5, 7-9, 13, 15-17, 20 and 22-24.

Rejections Under 35 USC §112, Second Paragraph

Claims 4-9, 12-17 and 20-24 were rejected under 35 USC §112, Second Paragraph, for indefiniteness. The Examiner asserts that dependent claims 4, 12 and 18 recite "wherein said mechanism is divided." In response, applicants have amended the claims to recite:

"wherein said computer system comprises a multiprocessor system, which comprises a plurality of processors, and said method is implemented in each of said processors independently." Accordingly, applicants respectfully assert that the rejection of claims 4, 12 and

18 (as amended) under 35 USC §112, Second Paragraph is obviated, and request withdrawal of same claim rejections is requested.

Claims 5-9, 13-17 and 20-24 are rejected for reciting "the first code component" and/or "the new code component" without proper antecedent support. As mentioned above, applicants have amended all of the pending claims where necessary to change "first source code component" and "new source code component" to -- first code component -- and -- new code component.-- Accordingly, applicants respectfully request withdrawal of the rejection of claims 5-9, 13-17 and 20-24 under section 112, Second Paragraph.

Response To rejections Under 35 USC §102

Claim 1 was rejected under 35 USC §102(b) as unpatentable over US Patent No. 6,219,690 to Slingwine. The Examiner asserts that Slingwine discloses in a computer system using an operating system to provide access to hardware resources, wherein said operating system provides access to said resources via a first source code component, a method of replacing said first source code component with a new source code component while said operating system remains active and while said operating system provides continual availability to applications of the hardware resources (Fig. 2, kernel running, col. 10, lines 54-67). The Examiner asserts that Slingwine further includes identifying references to said first source code component (108, col. 6, lines 41-50, col. 9, line 45 to col. 10, line 17, Fig. 4, Fig. 6); and replacing the identified references to said first source code with references to said new source code component (110, Fig. 3, 90, Fig. 3, Fig. 4).

With respect to independent claims 10 and 18, the Examiner asserts the same rejections as asserted against claim 1.

In response, applicants respectfully assert that independent claims 1, 10 and 18 are patentable distinguishable from Slingwine for at least the following reasons.

Applicants' invention is directed to a computer system that uses an operating system to provide access to hardware resources via a first code component and replacing the first code

component with a new code component while the computer operating system remains active and while that operating system provides continual availability of the hardware resources to system applications. The method comprises the steps of identifying references to the first code component, and replacing the identified references to the first code component with references to the new code component, and swapping the first with the new code component. The swapping (hot swapping) is seamless to the hardware resources.

The invention in its various claimed embodiments perform a hot-swap, including that at establishing a *quiescent state* in the first code component, and identifying references to the first code component at the quiescent state, transferring the references to the first code component to the new code component, and then hot swapping the new code component safely and efficiently. The identified references are all of the outstanding references held by the clients to the first code component so that these references now point to the new code component. Efficiently achieving the operational state of the first code component (at quiescence) in the new code component is accomplished by tracking when threads associated with the first code component are created and destroyed. Quiescent state transfer of the references occurs via a best-negotiated protocol implemented by the invention between the first and new code components. Finally, using an object translation table, all calls to the first code component are rerouted to the new code component, in accordance with the invention.

Slingwine, as distinguished, discloses a substantially zero overhead mutual-exclusion apparatus and method that allow concurrent reading and updating data while maintaining data coherency. The data reading process executes the same sequence of instructions that would be executed if the data were never updated, but rather than depending exclusively on overhead imposing locks, the mutual-exclusion mechanism tracks a thread execution history to determine safe times for processing a current generation of data updates while the next generation of data updates is concurrently being saved. When the last thread passes a quiescent state, a summary of thread activity triggers a callback processor that it is safe to end the current generation of updates. The callback processor restarts the summary of thread activity and initiates a next generation of updates. All data-updating threads pass through a quiescent state between the time they attempt to update data and the time the data are actually updated.

Slingwine does not swap a first code component with a new code component as does applicants' invention as set forth in independent claims 1, 10 and 18.

While the Examiner asserts that Slingwine discloses in a computer system using an operating system to provide access to hardware resources, wherein said operating system provides access to said resources via a first source code component, a method of replacing said first source code component with a new source code component while said operating system remains active and while said operating system provides continual availability to applications of the hardware resources (Fig. 2, kernel running, col. 10, lines 54-67), applicants respectfully disagree.

That is, Slingwine defines a quiescent state for a thread as that time that the thread is not accessing their mutual-exclusion mechanism. The mutual-exclusion mechanism is shown in Fig. 4, and described at col. 10, lines 52-67. Slingwine's mutual-exclusion mechanism is not equivalent to applicants' claimed method (mechanism), and is not understood, at the cited col. 10 text, to describe, in a computer system using an operating system to provide access to hardware resources, wherein said operating system provides access to said hardware resources via a first code component, a method of replacing said first code component with a new code component while said operating system remains active and while said operating system provides continual availability to the hardware resources by applications operational in the computer system.

While the Examiner asserts that Slingwine further includes identifying references to said first source code component (108, col. 6, lines 41-50, col. 9, line 45 to col. 10, line 17, Fig. 4, Fig. 6), applicants again respectfully disagree. The text at col. 6, lines 41-50, describes Slingwine's callback processor (104), which causes a next generation of Slingwine callbacks to become a current generation of callbacks. It does not show applicants step of identifying references to the first code component, required by each of applicants' independent claims. The text at col. 9, line 45 through col. 10, line 17, summarizes Slingwine's summary of thread activity, its generation counter, its thread counter, and its callback processor that uses each of the thread bits, generation counter, thread counter and callback processor. The text does not show

applicants step of identifying references to the first code component, required by each of applicants' independent claims.

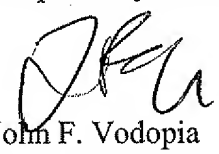
While the Examiner asserts that Slingwine further includes replacing the identified references to said first source code with references to said new source code component (110, Fig. 3, 90, Fig. 3, Fig. 4), applicants again respectfully disagree. Element 110 is a next generation and 90 is Slingwines' mutual exclusion mechanism. Neither can be said to applicants' replacing the identified references to the first code component with references to said new code component.

In view of the stated differences between amended independent claims 1, 10 and 18, taken as a whole, and Slingwine, applicants respectfully assert that independent claims 1, 10 and 18 are not anticipated by Slingwine under 35 USC §102(b), and respectfully request withdrawal of the rejections thereunder. Claims 2-9 depend from claim 1 and are patentable therewith; claims 11-17 depend from claim 10 and are patentable therewith; and claims 19-24 depend from claim 18 and patentable therewith. Applicant therefore respectfully asserts that claims 2-9, 11-17 and 19-24 are patentable in view of Slingwine Under Section 102 (b), and request withdrawal of same claim rejections in view of Slingwine.

The other references of record have been reviewed, and these other references, whether considered individually or in combination, also to not disclose or suggest the use of this invention as set forth in claims 1-24. The Examiner is, accordingly, respectfully asked to reconsider and to withdraw all of the objects to the Specification and claims, the rejections of claims 1-24 under 35 USC §112, Second Paragraph, and the rejection of claims 1-24 under 35 U.S.C. §103(a) in view of Slingwine, and to allow these claims.

If the Examiner believes that a telephone conference with applicants' attorneys would be advantageous to the disposition of this case, the Examiner is asked to telephone the undersigned.

Respectfully Submitted,



John F. Vodopia
Registration No.: 36,299
Attorney for Applicants

Scully, Scott, Murphy & Presser. P.C.
400 Garden City Plaza, Suite 300
Garden City, New York 11530
(516) 742-4343

JFV:tb